A Hybrid Approach for Proving Noninterference and Applications to the Cryptographic Verification of Java Programs

Ralf Küsters*, Tomasz Truderung*, Bernhard Beckert†, Daniel Bruns†, Jürgen Graf† and Christoph Scheben†
*University of Trier, Germany †Karlsruhe Institute of Technology, Germany

I. Introduction

The problem of checking noninterference properties of programs has a long tradition in the field of computer security and, in particular, in language-based security. A program is called noninterferent (w.r.t. confidentiality) if no information from high variables, which contain confidential information, flows to low variables, which can be observed by the attacker or an unauthorized user. Several tools and approaches exist in the literature for checking noninterference. Some approaches, such as type checking, abstract interpretations, and program dependency graphs, with tools including Joana, JIF, and TAJ, have a high degree of automation, but they overapproximate the actual information flow, and hence, may produce false positives. Others, such as those based on theorem proving, with tools such as KeY, Isabelle, and Coq, allow for very precise analysis, but need human interaction and, hence, analysis is often time-consuming

Certainly, fully automated tools are preferable over interactive approaches. However, if automated tools fail due to false positives and the analysis cannot further be refined by these tools, because, for example, the tools do not allow this or run into scalability problems, the only option for proving noninterference so far is to drop the automated tools altogether and instead turn to fine-grained but interactive, and hence, more time-consuming approaches, such as theorem proving. This "all or nothing" approach is unsatisfying and problematic in practice.

In this paper, we therefore propose a simple, tool-independent approach, which we call *hybrid approach* and which allows one to use automated analysis of noninterference properties as much as possible and only resort to more fine-grained analysis at places in a program where necessary, where the latter analysis requires checking merely specific functional properties in parts of the program, rather than checking the more involved noninterference properties (for the whole program).

While our hybrid approach should be widely applicable—it is not tailored to specific tools or specific applications, and the basic idea is quite independent of a specific programming language—, our main motivation comes from the problem of checking, on the implementation level, cryptographic properties of programs (that use cryptography), where here we consider Java programs. This has become an active field of research in the last few years. In this paper, besides the hybrid approach, we make contributions also to this problem, which are of independent interest. More precisely, the contributions of this paper are as follows.

Our hybrid approach is stated and proven for the language Jinja+, a rich fragment of Java. The basic idea underlying this approach is as follows: Given a program P, we first run an automated tool on P. If this fails due to (what we think are) false positives, we add some code, following rules of our approach, to P at places where the tool has problems, to make it more explicit and more clear for the automated tool that there is no illegal information flow. A typical case is that we add an assignment to a variable with an expression that makes explicit that this variable does not depend on high input. (Formulating the expression may require to gather some data at other places in a program.) It might be necessary to iterate this process until the tool does not produce false positives. Let P' denote the resulting extension of P and assume that the automated tool showed that P' has the desired noninterference property. Now, we need to show that P' is what we call a *conservative* extension of P'. This basically means that P and P' behave the same, i.e., the additional code did not change the behavior of the original program. In particular, if an assignment was added, then right before the execution of the assignment the variable should already have the value that is then assigned to it. In other words, the assignment is redundant. Proving that an extension is conservative would now typically require some more precise and possibly interactive tool. However, the analysis should typically be restricted to certain fragments of the program (namely parts in which the automated tool had problems) and involve merely the analysis of specific functional properties, rather than checking the more intricate noninterference properties (for the whole program). The key property that we show for the hybrid approach to work is that if P' is noninterferent and is a conservative extension of P, then P is noninterferent as well. To the best of our knowledge, this seems to be a new approach for proving noninterference.

As mentioned, our hybrid approach should be widely applicable. The basic concept is quite independent of a specific programming language. Also, the approach is not tailored to specific tools or applications.

The application domain for the hybrid approach we are mainly interested in is the problem of checking cryptographic indistinguishability properties for Java programs. In [3], a framework was developed that enables tools that can check (standard) noninterference properties for Java programs, but a priori cannot deal with cryptography (probabilities, polynomially bounded adversaries), to establish cryptographic indistinguishability properties of Java programs. The framework combines techniques from program analysis and universal composability. Given a Java program (that uses cryptography), the idea is to first check noninterference for this program where cryptographic operations (such as encryption) are performed within so-called ideal functionalities, in the sense of universal composability. The framework then guarantees that the actual Java program, where the ideal functionalities are replaced by the actual cryptographic operations, enjoys cryptographic indistinguishability properties.

In a case study, we use the hybrid approach and the mentioned framework to establish cryptographic privacy properties for a simple e-voting system implemented in Java. In this system, voters can send their votes over a confidential and authenticated channel (realized using public-key encryption and signatures) to a server which, after the voting phase is finished, calculates the result of the election and posts it, using an authenticated channel (realized using signatures), on a bulletin board. Everybody can then obtain the result of the election from the bulletin board. Since we are interested in checking privacy of votes for this system, in a set-up phase, the adversary can provide two vectors of votes for honest voters. It is checked whether these vectors result in the same election outcome, i.e., whether the number of votes for each candidate is the same. If this is not the case, the system aborts. Otherwise, honest voters vote according to one of the vectors provided by the adversary. Which vector is chosen depends on a secret (high) bit. Now, the system provides privacy if the adversary cannot distinguish which of the two vectors was chosen. In other words, secrecy of the high bit is preserved, a property which, based on the mentioned framework, we would like to establish by an automated tool for checking noninterference. In our case study, we use the fully automated tool Joana [2] for this task.

The problem is that Joana produces a false positive (and probably all other automated tools would do this for our e-voting system). Joana cannot see that the publication of the election outcome does not constitute an illegal information flow. Roughly, the reason that there is no information leakage is that the election outcome is determined by the vectors provided by the adversary, which constitute low input and induce the same election outcome. So provided that the server calculates the correct result, it should in fact correspond to the result induced by the vectors. Now, in order to see that there is no illegal flow, Joana would have to verify that the server calculates and outputs the correct result, namely the one induced by the vectors. This is beyond what Joana can do.

Using our hybrid approach, in combination with Joana and the theorem prover KeY [1], we can nevertheless establish the desired property for our system, where KeY needs to prove only the functional property that the server correctly calculates and outputs the election result. The fact that otherwise the clients, the server, and the bulletin board do not leak secret information is established by Joana. We note that the analysis with KeY is mostly finished but is ongoing work.

In order to apply the mentioned framework to this case study — by which we obtain cryptographic guarantees by verifying (standard) noninterference properties —, we need to provide an ideal functionality for secure message transmission, i.e., confidential and authenticated message transmission, as well as an ideal functionality for authenticated message transmission, and we need to show that these functionalities can be realized using standard (IND-CCA2-secure) encryption schemes and (EU-CMA-secure) signature schemes. By the framework, it then suffices to verify noninterference of our e-voting system when it uses these functionalities instead of the actual cryptographic schemes.

These ideal functionalities and their realizations are of general interest. They can be used beyond this particular case study to establish cryptographic indistinguishability properties for Java programs that use such primitives. They also further instantiate the framework of [3], for which so far only an ideal functionality for public-key encryption has been considered. As such they constitute another important contribution of our work. The proofs of the realizations of these functionalities are non-trivial. They are carried out in a modular way. While in the cryptographic literature similar functionalities and their realization have been considered before in a Turing machine model, here these functionalities are formulated in Java and they can actually be used in Java programs. This requires some care. In addition, the proofs are carried out with respect to Jinja+ semantics.

References

[1] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth, Steffen Schlager, and Peter H. Schmitt. The KeY tool. *Software and System Modeling*, 4:32–54, 2005.

[2] Christian Hammer and Gregor Snelting. Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. *Int. J. Inf. Sec.*, 8(6):399–422, 2009.

[3] Ralf Küsters, Tomasz Truderung, and Jürgen Graf. A Framework for the Cryptographic Verification of Java-like Programs. In *IEEE Computer Security Foundations Symposium, CSF 2012*. IEEE Computer Society, 2012. To appear.